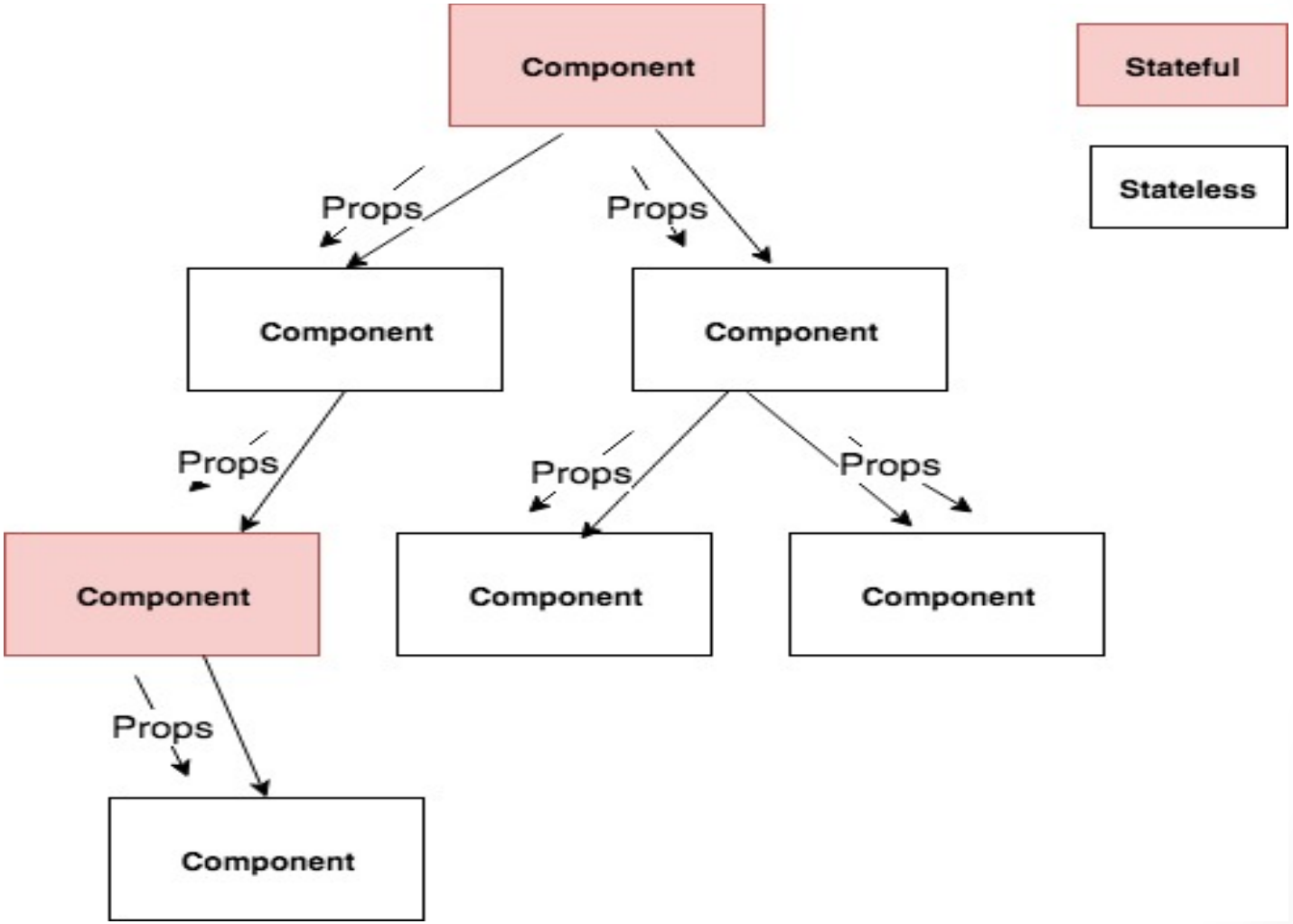
ReactJS.

The basics (continued)

Summary to date

- React focused on building web Uis.
- The component is the core building unit.
- A React app is designed as a herarchy of components.
- JSX - a declarative language for describing a component's UI
  - A HTML-like syntax.
  - Allows embedded JS expressions.
- Component data: Two types,
  1. Props - passed in; immuttable
  2. State – managed internally mutates as a result of some event.
- State data changes cause:
  1. The component to rerender.
  2. Subordinate components also render, with updated props
- React uses a virtual DOM to manage real DOM updates.

Unidirectional data flow

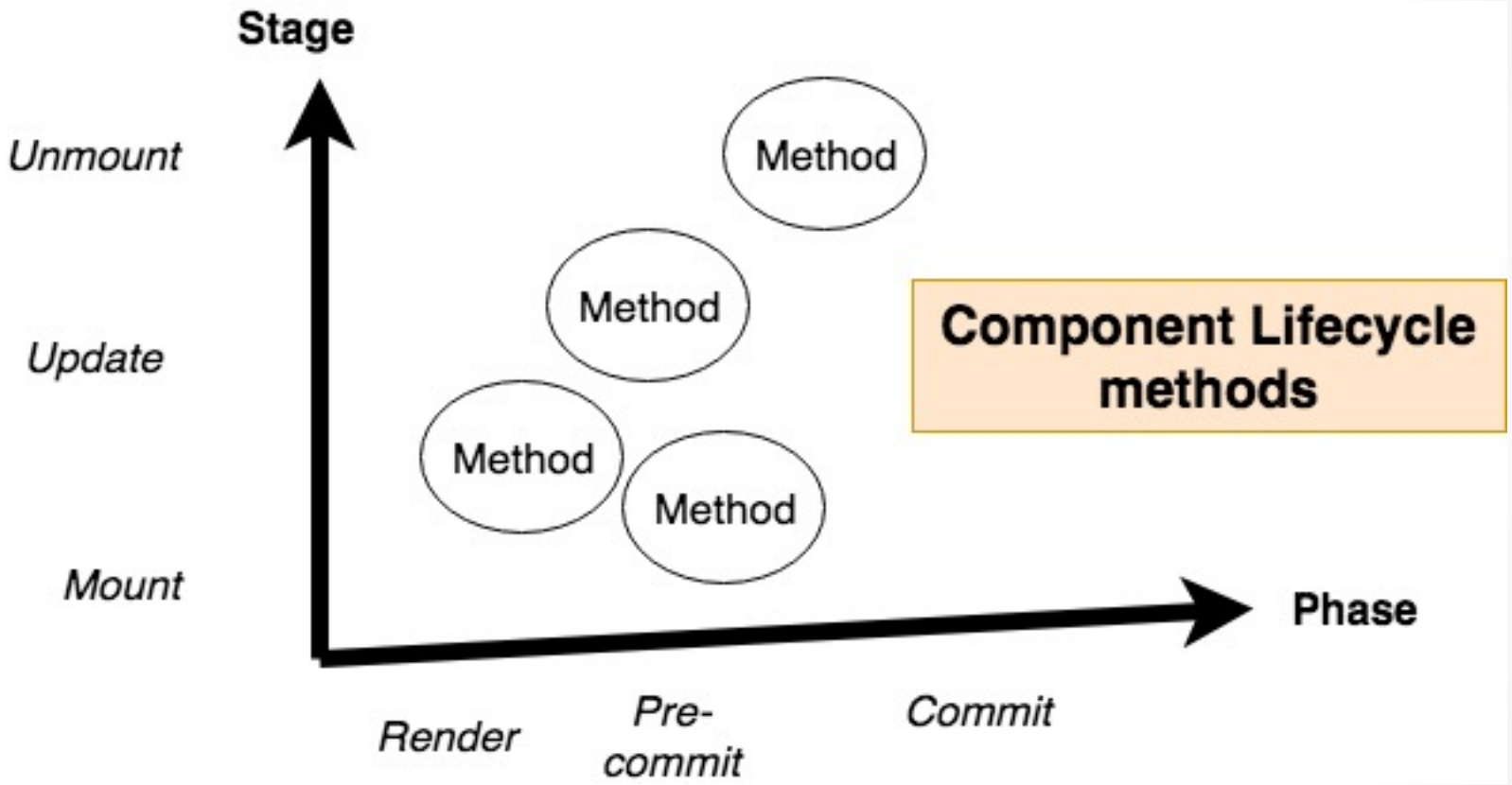


Unidirectional data flow

- **In a React app, data flows unidirectionally ONLY.**
  - **Most other SPA frameworks use two-way data binding.**

- **In a multi-component app, a common pattern is:**
  - **A small subset (naybe only 1) of components will be statefull – the majority will be stateless.**
  - **Statefull components manage:**
    - **Passes any state changes to subordinate components via props, If necessary.**
    - **Calls *setState*() to update its state.**
      - *React guarantees subordinate components are re-rendered with new prop values.*

Component *Lifecycle methods*



Component *Lifecycle methods*

- **Methods invoked by React at specific times in a component's lifecycle (Most are optional).**

- **Lifecycle stages:**
  1. **Mounting (Initialization).**
  2. **Update.**
     a) **New props.**
     b) **setState();.**
     c) **forceUpdate.**
  3. **Un-mounting.**

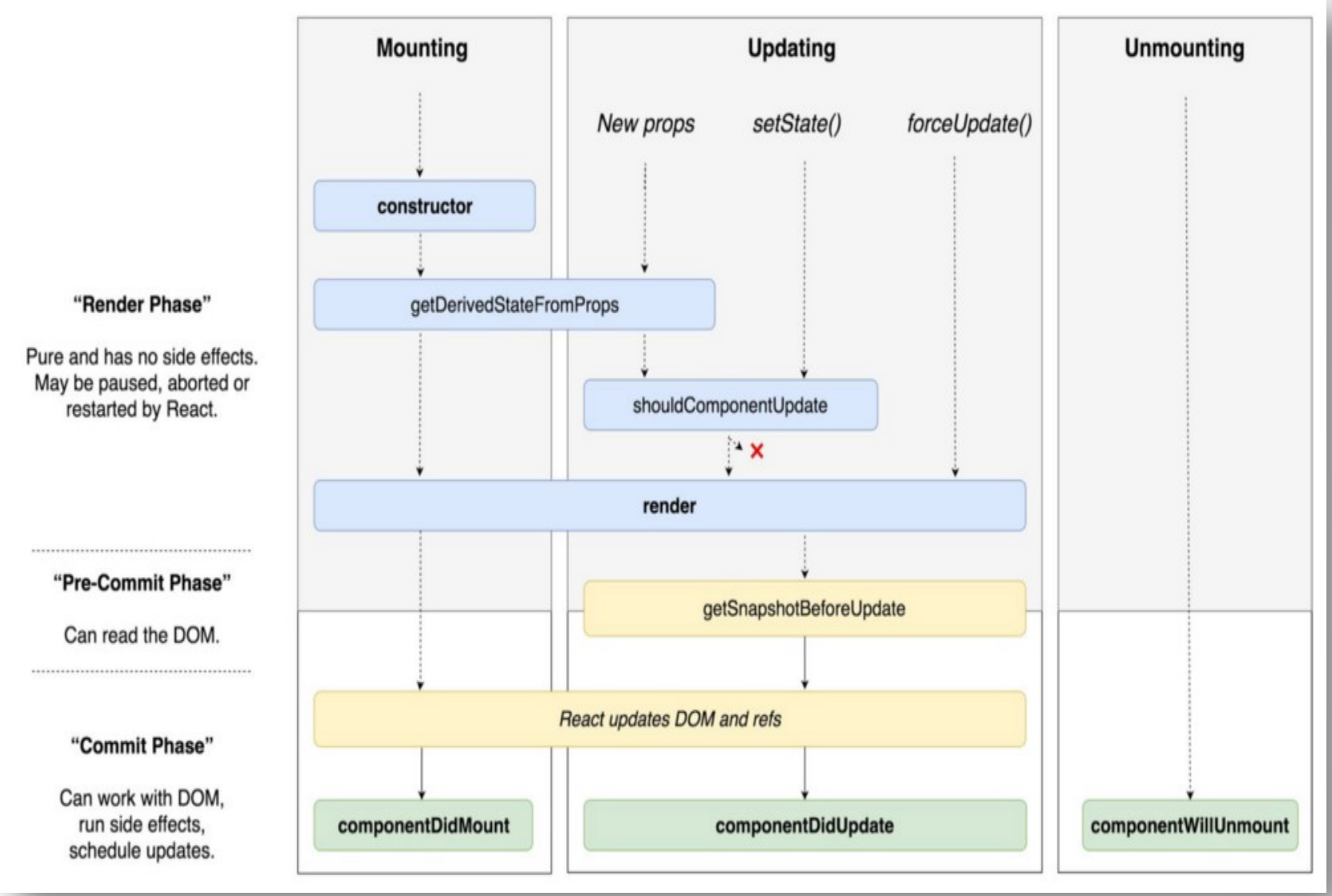
- **Phases:**
  - **Render phase.**
  - **Pre-commit phase (Pre DOM update)**
  - **Commit phase.**

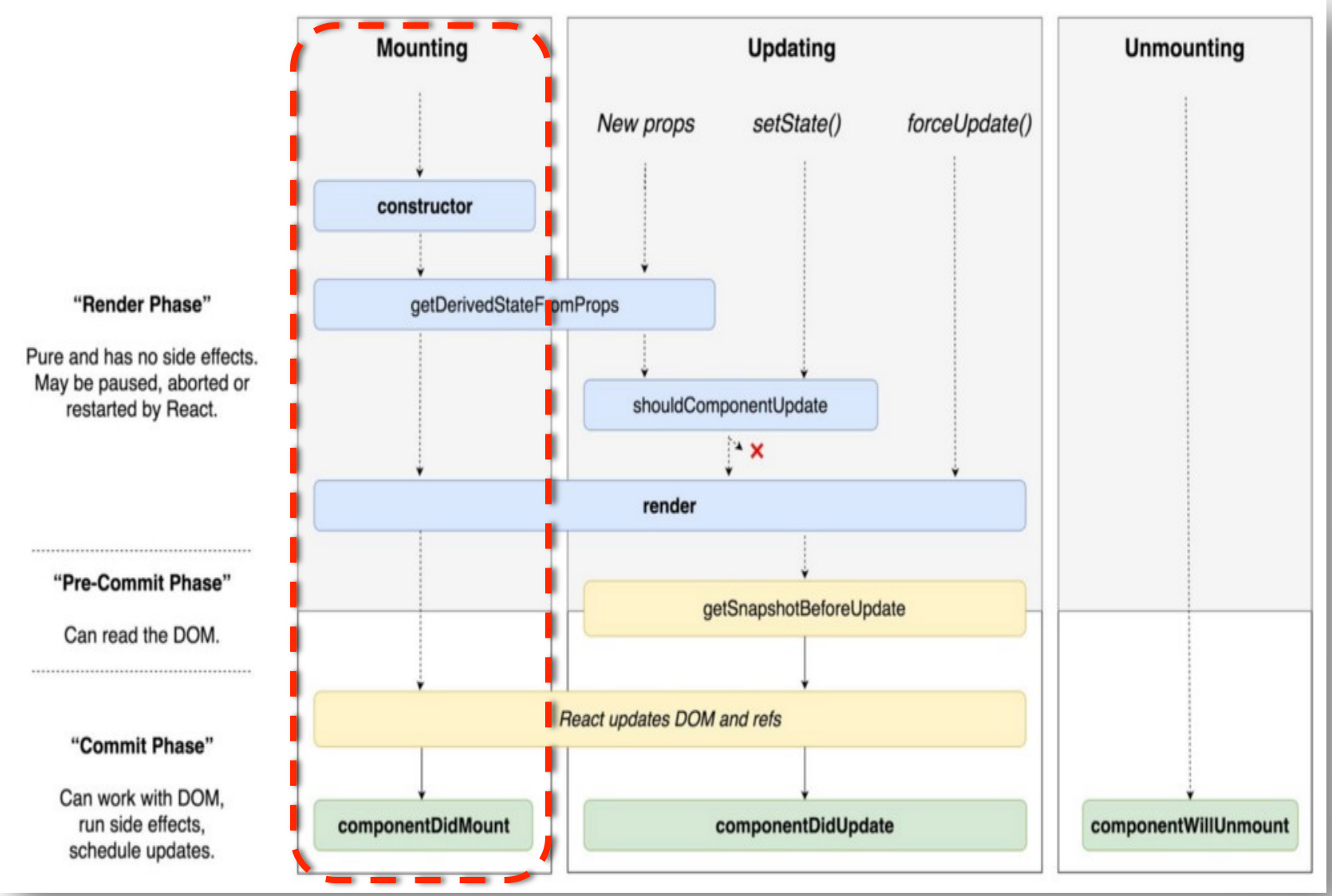
The Lifecycle methods

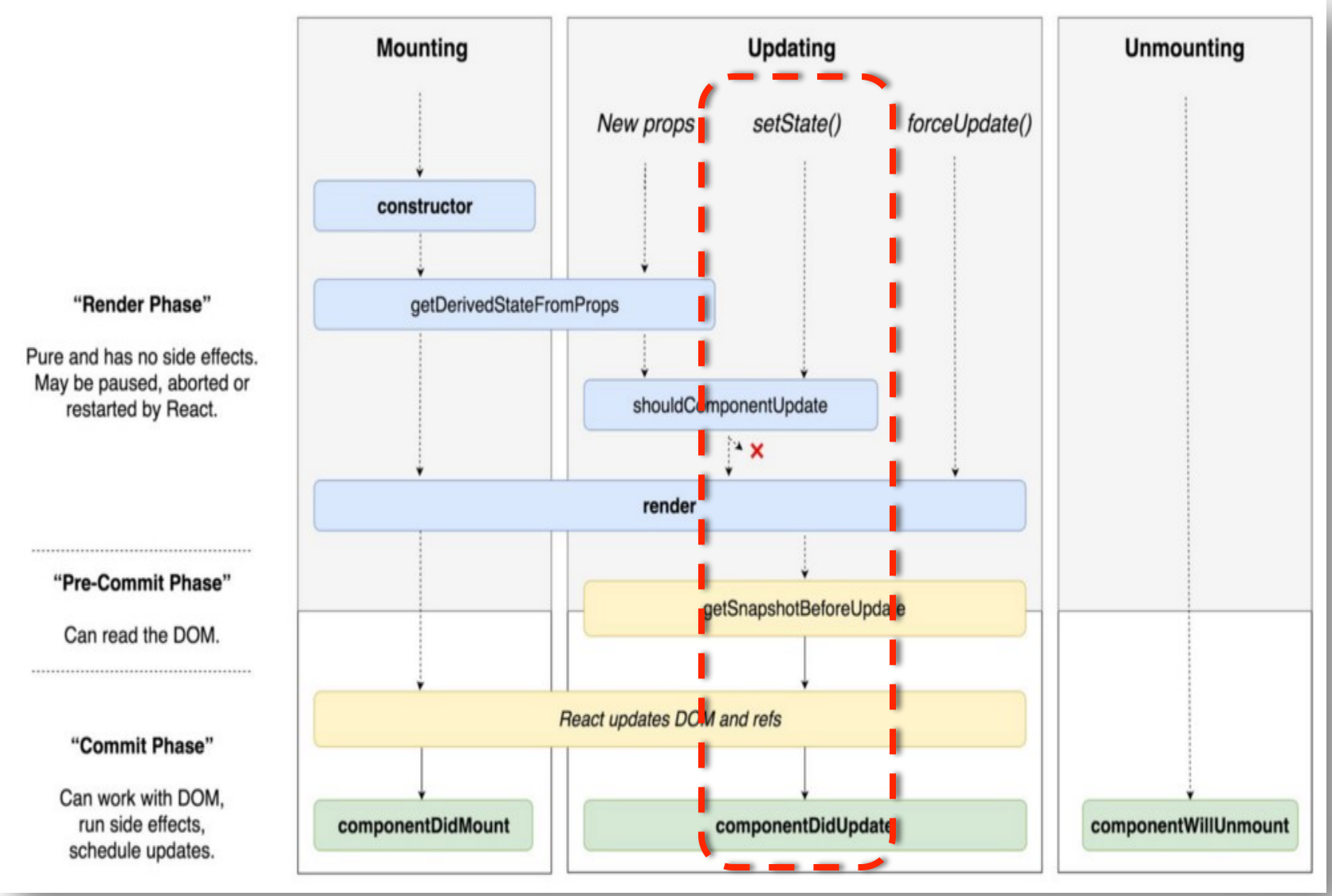
- shouldComponentUpdate**() – returns boolean – can cause a component to skip re-rendering.**
- getDerivedStateFromProps**() - when a component state object is computed from its prop values.**
- componentDidUpdate**() – executed after a rendering has updated real DOM; used to perform real DOM manipulatio, e.g. set up event handler or cause side-effect.**
- componentDidMount**() – executed once, after component has mounted (see later)**
- **componentWillUnmount(); executed before a component is about to unmount; Perform cleanup operations, e.g. remove DOM event handler.**

The Lifecycle methods

- *shouldComponentUpdate()* ✔
- *getDerivedStateFromProps()*
- *.*
- *render().* ✔
- *componentDidUpdate()*
- *componentDidMount()* ✔
- *componentWillUnmount***()**


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

Sample App

- Filtered Friends App.
- Component hierarchy (App Design:

  FriendsApp

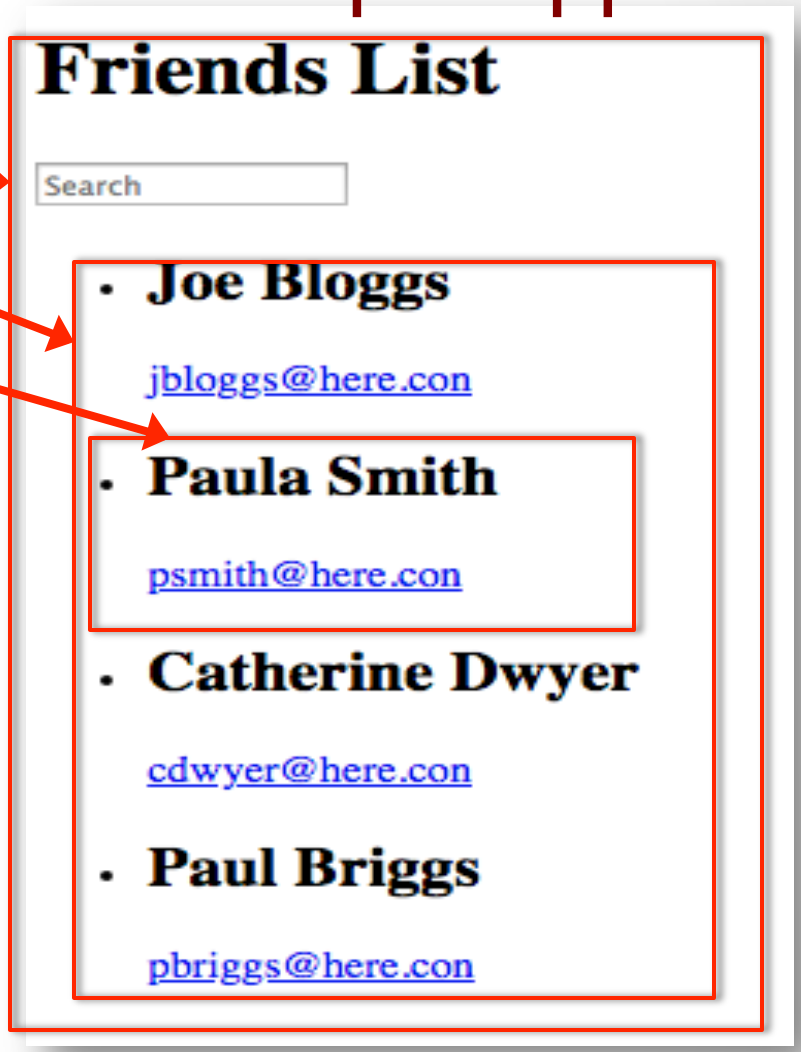
  → *FilteredFriendList*

  → *Friend*

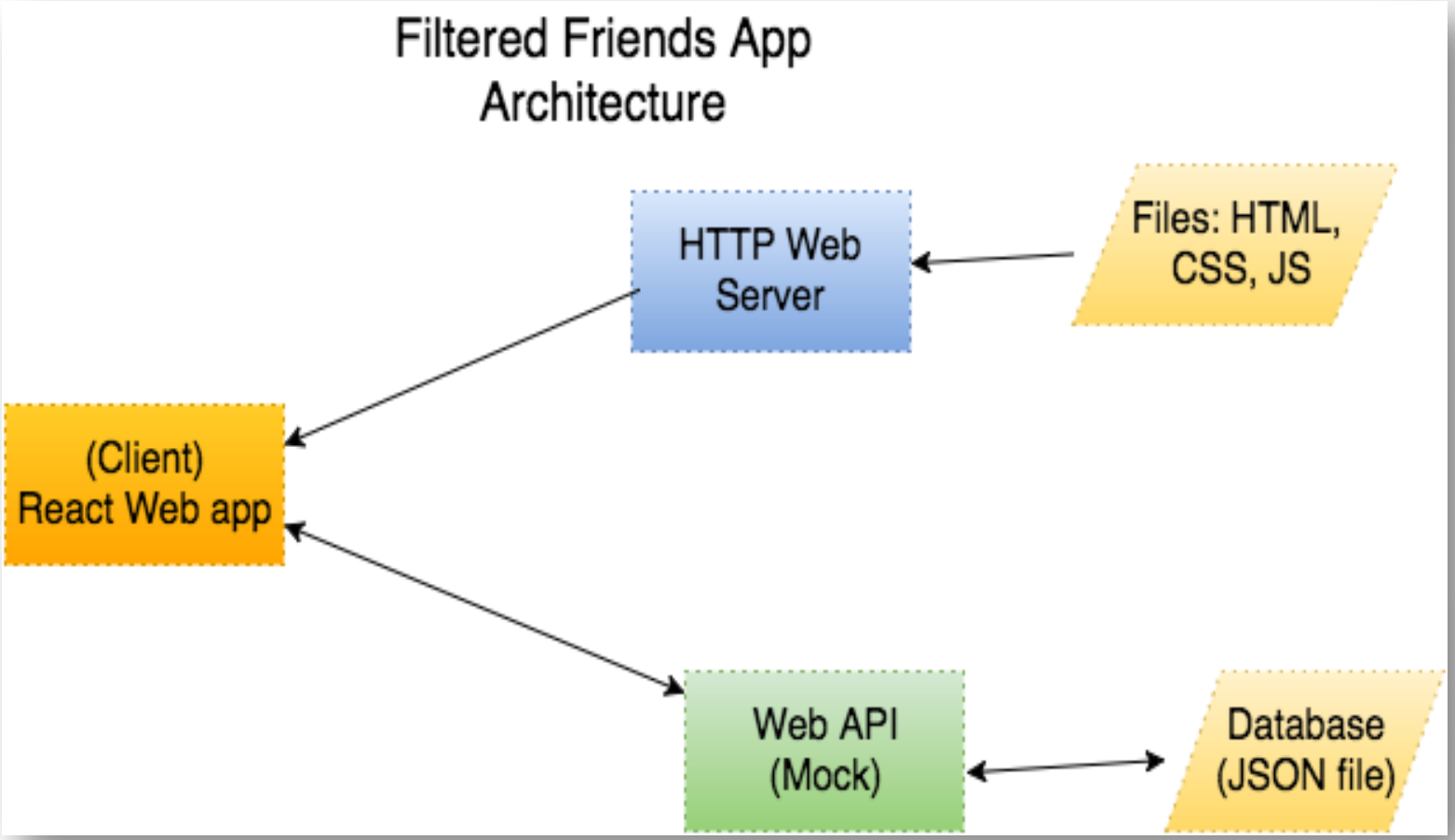
FriendsApp component:

1. Manages app's state (i.e. text box content).
2. Computes matching friends list.
3. Controls list re-rendering.

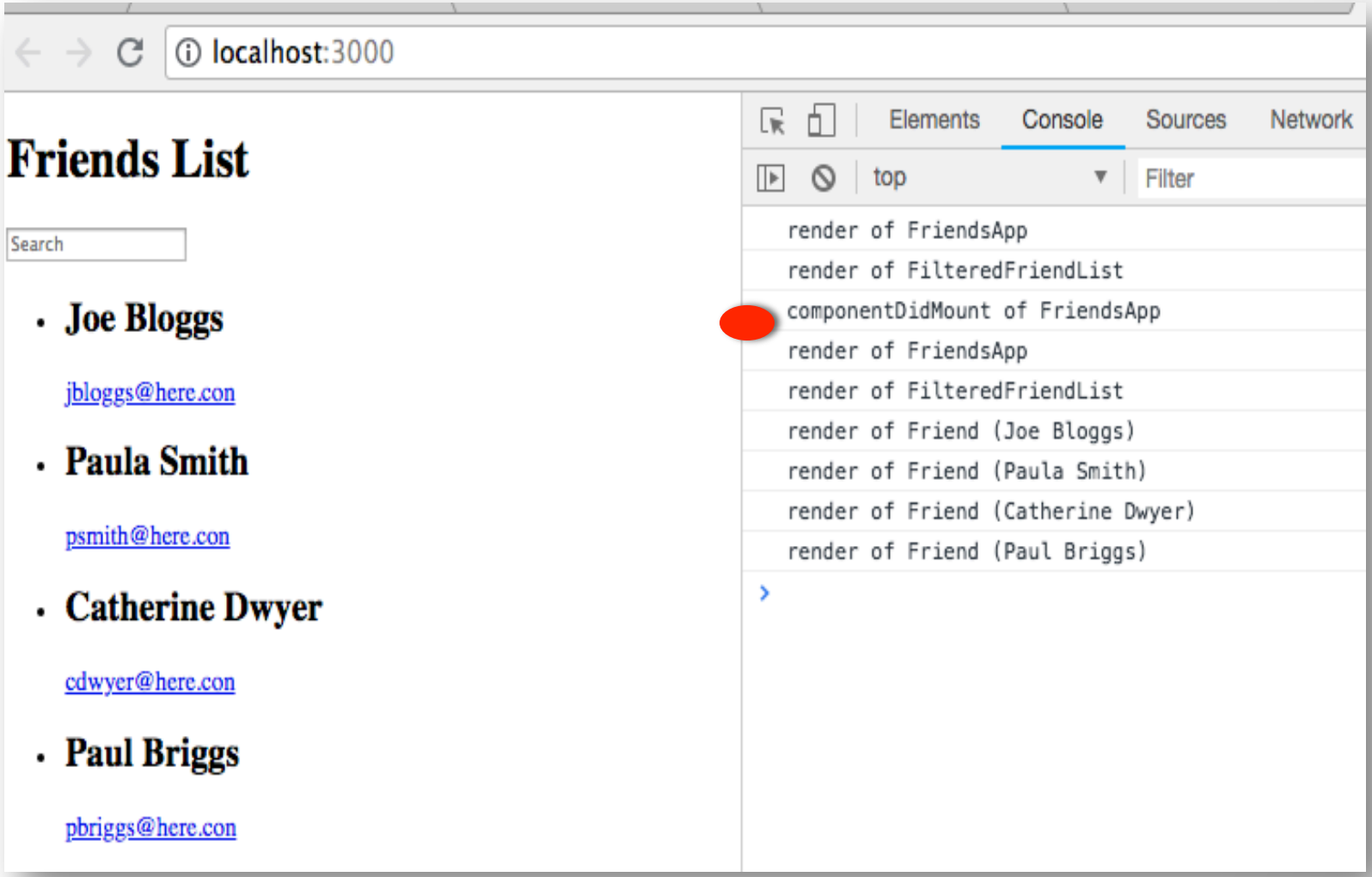
- [Alternative design later.]

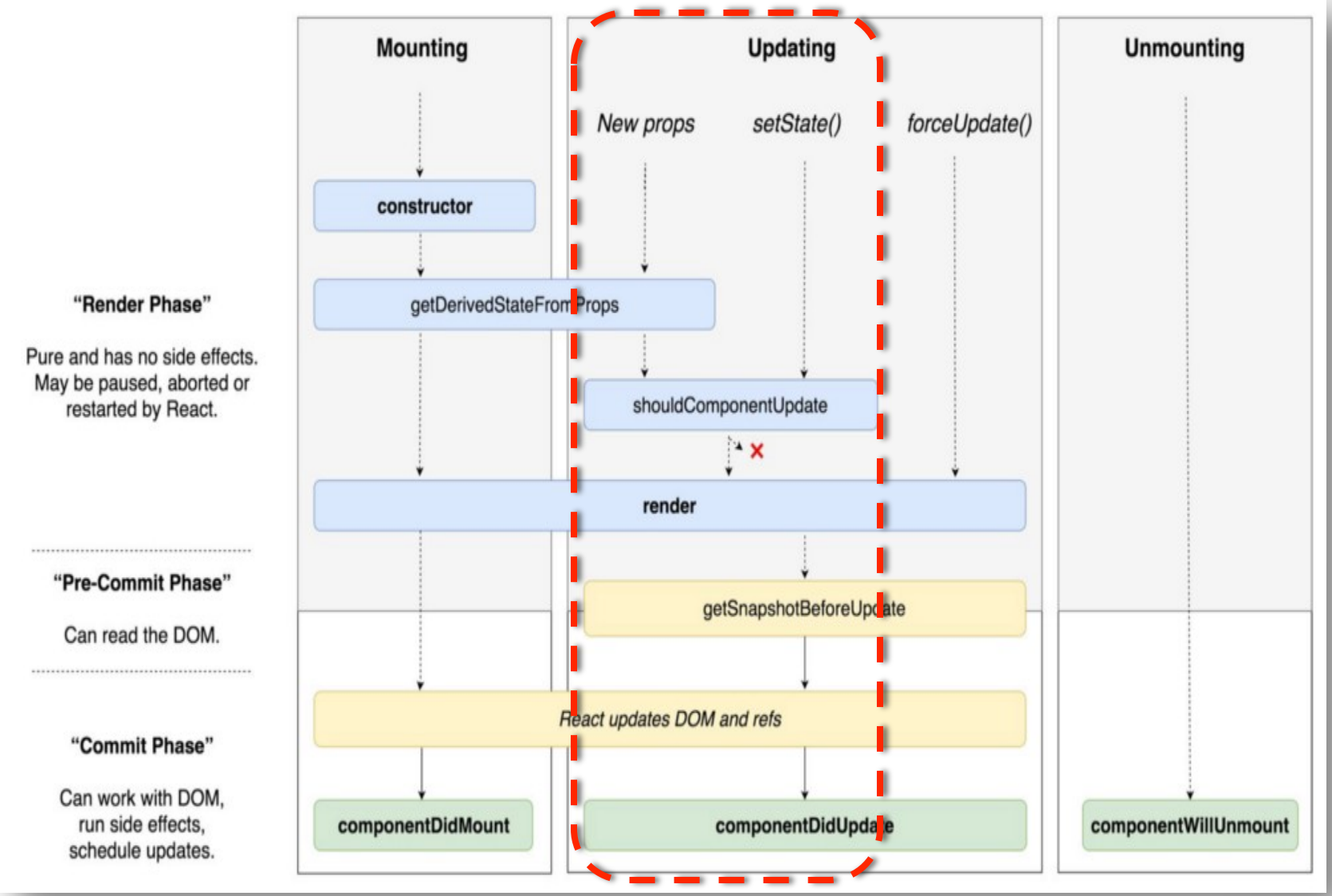


Sample App – Architecture..

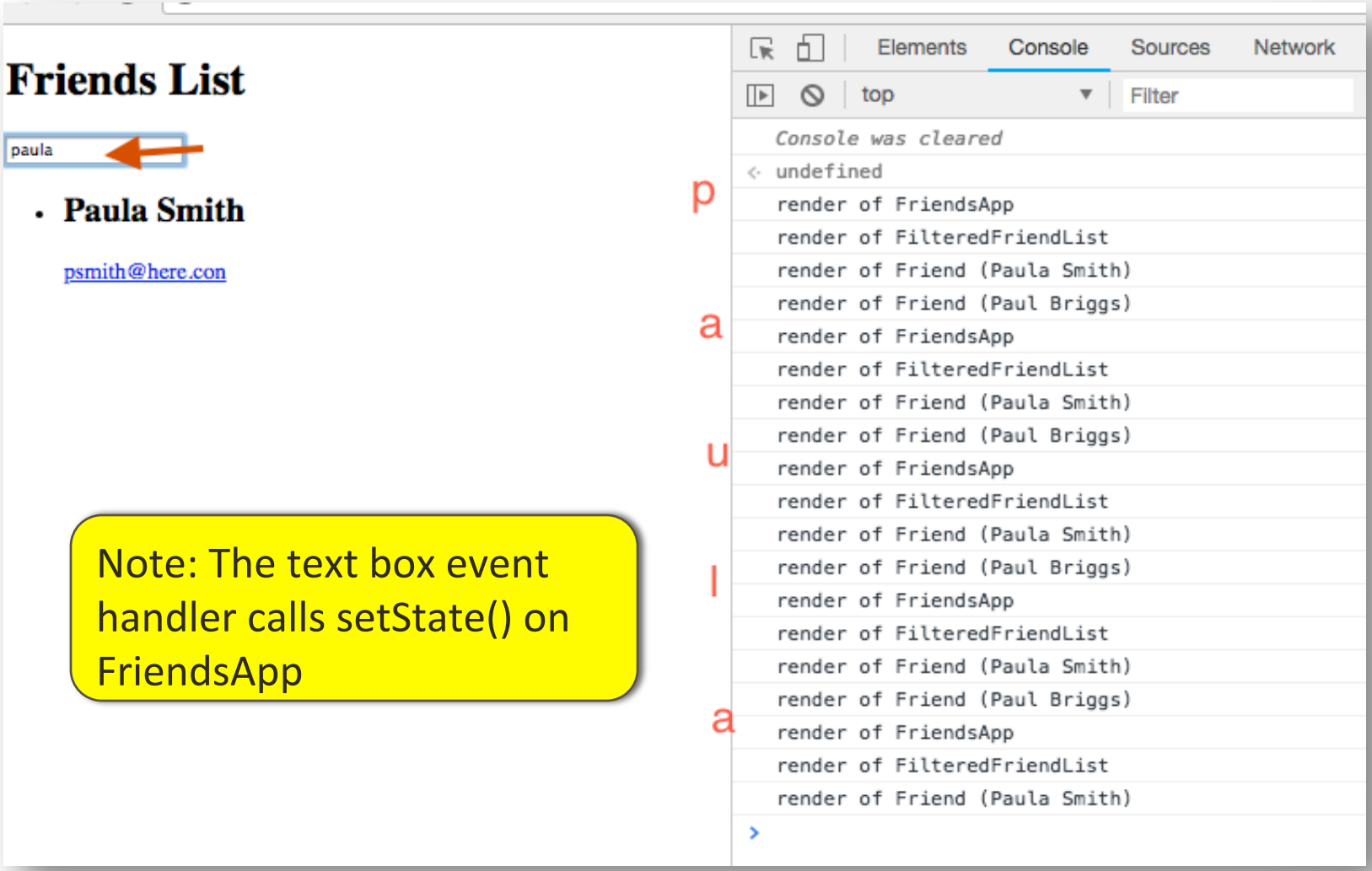


Sample App – Execution trail (Mounting & setState)..




Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

Sample App – Execution trail (Update on new props & setState)..

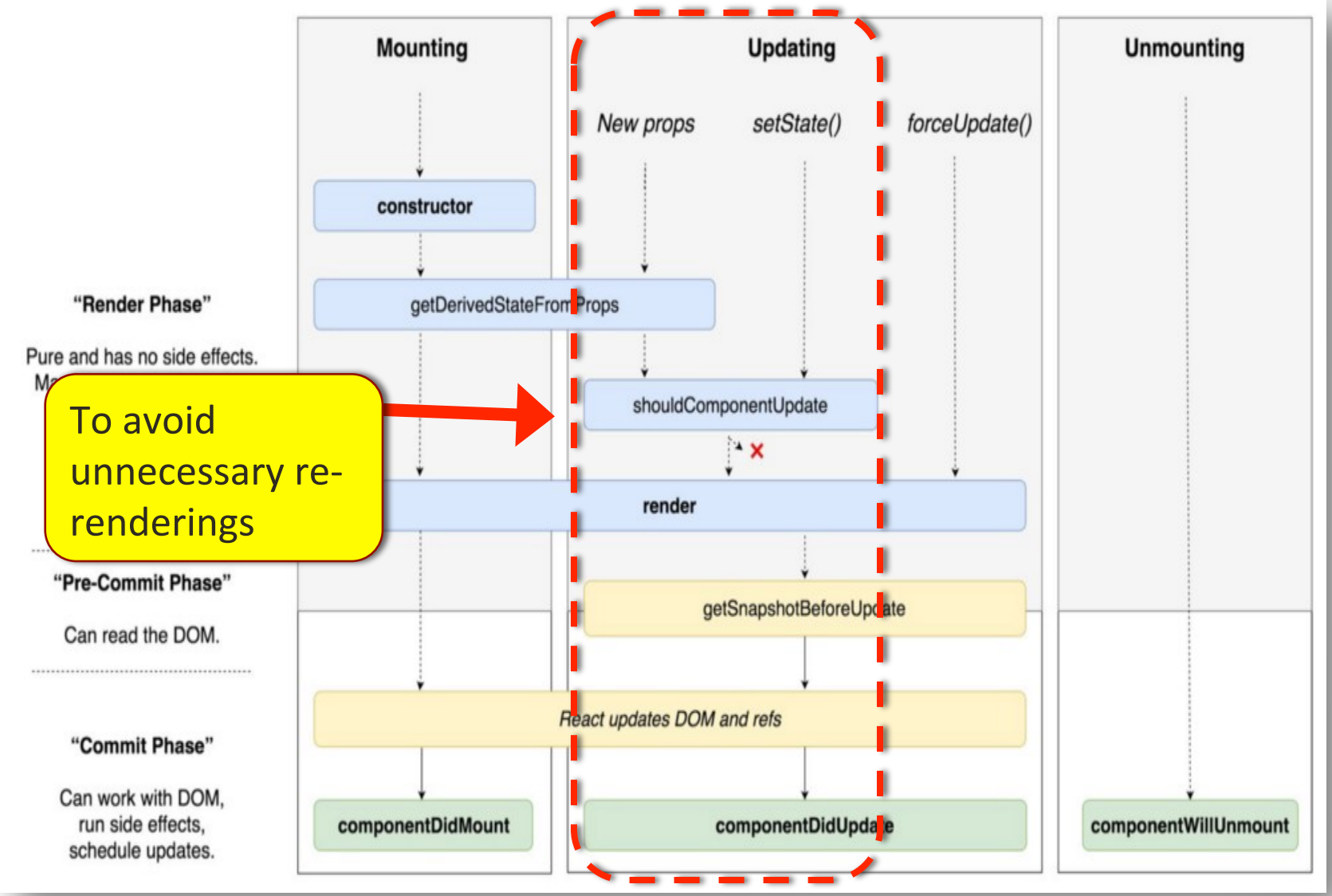


Note: The text box event handler calls setState() on FriendsApp

Unidirectional data flow & **Re-rendering**

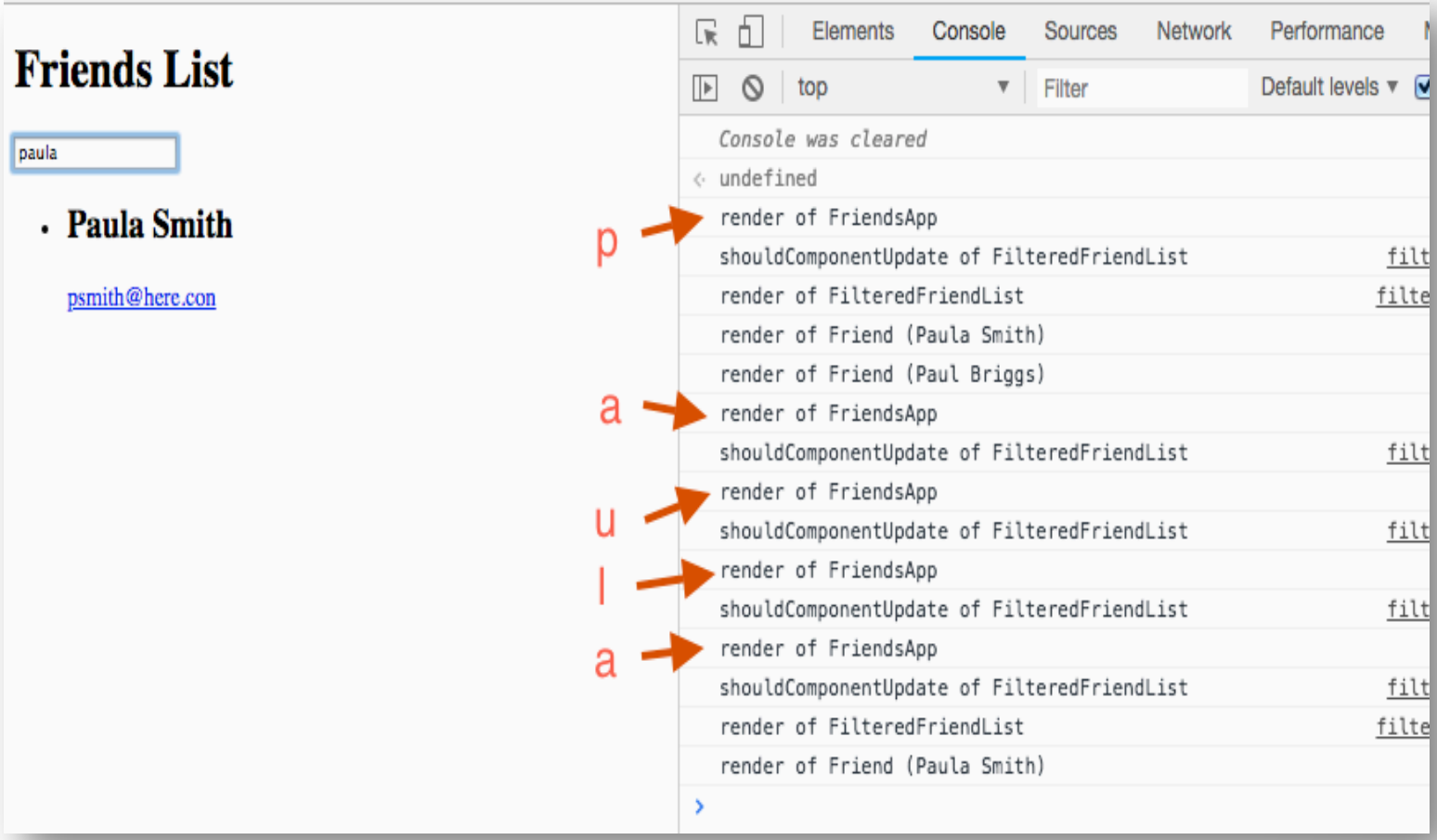
- **What happens when user types in to text box?**

  User types a character into text box
  → *onChange* ***event handler executes***
  - → ***Handler calls setState() (****FriendsApp* ***component)***
    - → ***React calls*** *FriendsApp* ***render() method***
    - → ***React calls render() method of children (****FilteredFriendList****) with new prop values***
    - → ***React calls render() method of*** *FilteredFriendList* ***children.***
    - → ***(Pre-commit phase) React re-computes the new Virtual DOM***
    - → ***React diffs the new and previous Virtual DOMs***
    - → ***(Commit phase) React batch updates the Real DOM***
    - → ***Browser repaints screen***


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
To avoid unnecessary re-renderings

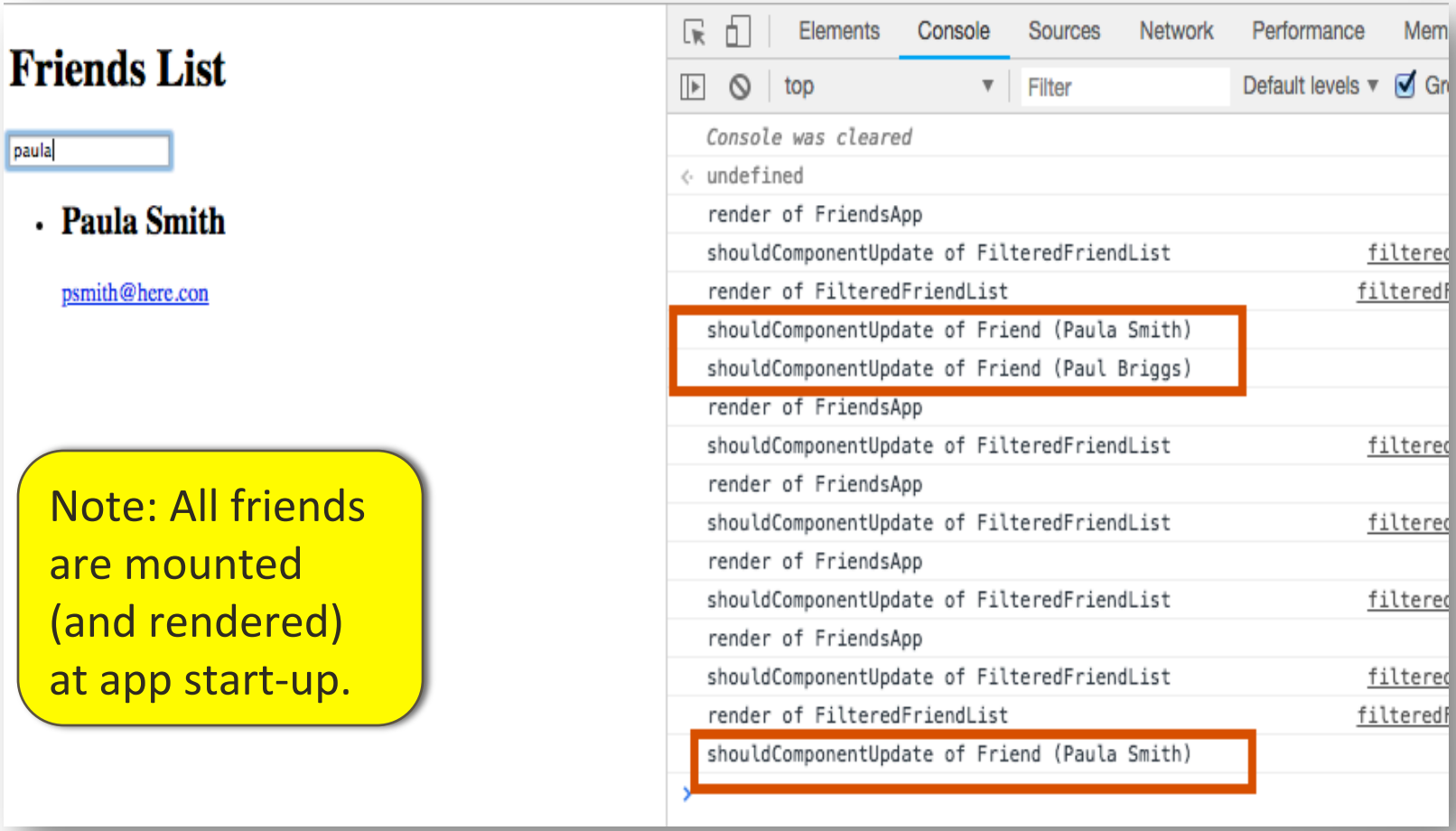
Sample App – Execution trail (Update on new props & setState)..

FilteredFriendsList should NOT re-render if the the length of array prop (of matching friends) has not changed

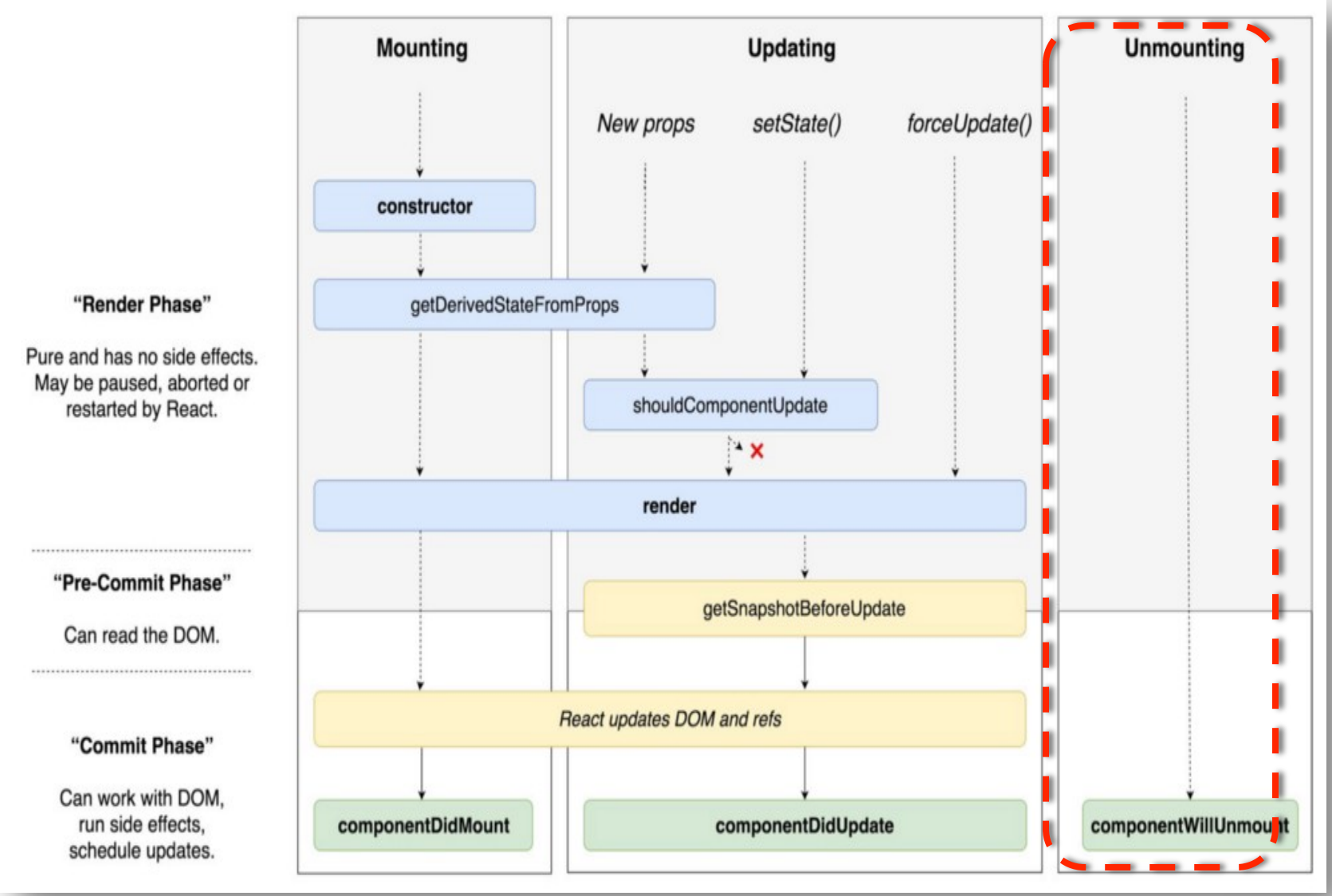


Sample App – Execution trail (Update on new props & setState)..

Friend should NOT re-render once it is mounted



Note: All friends are mounted (and rendered) at app start-up.


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

Sample App – Execution trail (Un-mounting)..

```
render of FriendsApp
render of FilteredFriendList
componentDidMount of FriendsApp
render of FriendsApp
shouldComponentUpdate of FilteredFriendList
render of FilteredFriendList
render of Friend (Joe Bloggs)
render of Friend (Paula Smith)
render of Friend (Catherine Dwyer)
render of Friend (Paul Briggs)
render of FriendsApp
shouldComponentUpdate of FilteredFriend
render of FilteredFriendList
shouldComponentUpdate of Friend (Paula Smith)
shouldComponentUpdate of Friend (Paul Briggs)
componentWillUnmount of Friend (Joe Bloggs)
componentWillUnmount of Friend (Catherine Dwyer)
render of FriendsApp
shouldComponentUpdate of Filtere
render of FilteredFriendList
render of Friend (Joe Bloggs)
shouldComponentUpdate of Friend (Paula Smith)
render of Friend (Catherine Dwyer)
shouldComponentUpdate of Friend (Paul Briggs)
>
```

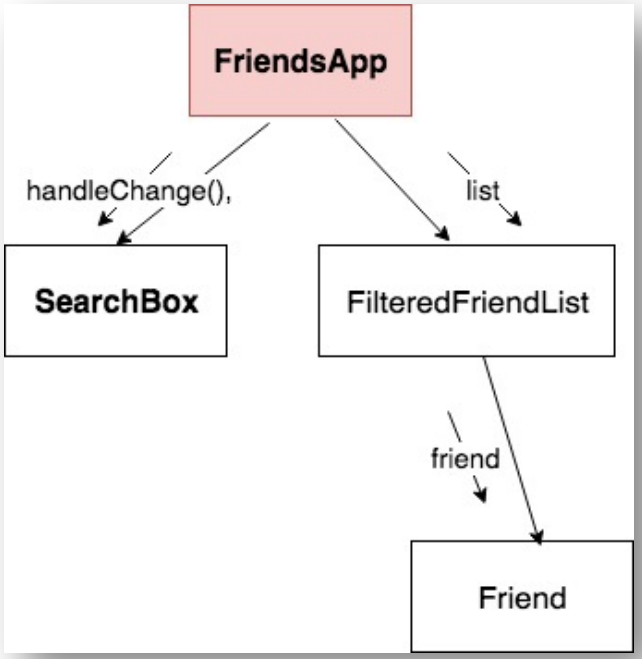
App start-up

Typed ‘p’

Typed ‘<del>’

Inverse data flow

- **What if a component's state is effected by an event in a subordinate component?**
- **Solution: The** inverse data flow pattern.



Aside – ES6 destructing

- **Fragment (destruct) a collection structure (arrays and objects).**

```
let nums = [0, 1, 2];
//    without destruction
let zero = nums[0];
let one = nums[1];
let two = nums[2];

//     with destruction
let [zero, one, two] = nums;
console.log(one);   // 1
```

```
let obj =
     { key1: 100, key2: 'hundred'};
//      without
let key1 = obj.key1;
let key2 = obj.key2;

// with
let {key1, key2} = obj;
console.log(key2) // hundred
```

Stateless Functional components

- **Many components only require the** render **method.**
- **The lifecycle methods are redundant but still effect performance.**
- **Use stateless functional components (sfc) where possible.**

```
const ComponentName = (props) =>
    { …. body of render method …..}
```

- Legacy code - jscodeshift **tool <u>transforms</u> conventional (class-based) components to sfc.**

$ npm install –g jscodeshift

….. Must also install transformer(s) seperately

$ jscodeshift -t transforms/pure-component.js --useArrows=true --destructuring=true <path to source file>

Stateless Functional components

```
class DynamicLanguages extends React.Component {
    render() {
        return (
            <div className='myCSSstyle' >
                <h1>{this.props.heading}</h1>
                <ul>
                    <li>{this.props.languages[0]}</li>
                    <li>{this.props.languages[1]} </li>
                </ul>
            </div>
        );
    }
}
// This component's props are heading and languages
```

Stateless Functional components

```
const DynamicLanguages = (  { heading, languages} ) => {
    return (
        <div className='myCSSstyle' >
            <h1>{heading}</h1>
            <ul>
                <li>{languages[0]}</li>
                <li>{languages[1]} </li>
            </ul>
        </div>
    );
};
```

3 Lab apps DEMO

JSON

JavaScript Object Notation

JSON

JavaScript Object Notation (JSON)

- A standard for serializing data into text form.
  - Alternative to XML serialization.

- Advantages:
  1. Human-readable (like XML, but easier).
  2. Useful for data interchange between applications (like XML, but less verbose).
  3. Useful for representing and storing semi-structured data.
     - Unlike the Relational data model, which is only suited for structured data.

- JSON is no longer tied to JavaScript – lots of languages have JSON parsers.

JSON

- JSON constructs:
  1. Base Values:
     - number, strings (double quoted), boolean (true / false), null.
  2. Composite values:
     a. Objects: enclosed in { } and consist of set of key-value pairs.
        - Key-value pair termed a <u>property</u>.
     b. Arrays: enclosed in [ ] and are lists of values.
     - Objects and arrays can be <u>nested</u>.

Safari File Edit View History Bookmarks Develop Window Help (24%) Tue 7:51

http://localhost:3000/weblogs

http://localhost:3000/weblogs

Google

Apple Yahoo! Google Maps Wikipedia News (570) Popular ToBeRead

https://people.ok.ubc.ca/rlawrenc... http://localhost:3000/weblogs


```
[
  {
    "created_at": "2013-01-26T15:24:23Z",
    "description": null,
    "id": 2,
    "title": "Sample 2",
    "updated_at": "2013-01-26T15:24:23Z"
  },
  {
    "created_at": "2013-01-26T15:24:23Z",
    "description": null,
    "id": 3,
    "title": "Sample 3",
    "updated_at": "2013-01-26T15:24:23Z"
  },
  {
    "created_at": "2013-01-26T15:26:08Z",
    "description": "bla bla bla",
    "id": 4,
    "title": "Software Micro-Frameworks",
    "updated_at": "2013-01-26T17:44:16Z"
  }
]
```

A Property.

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ],
  "Magazines":
  [
    { "Title":"National Geographic",
      "Month":"January",
      "Year":2009 }
    ,
    { "Title":"Newsweek",
      "Month":"February",
      "Year":2009 }
  ]
}
```

Semi-structured

Relational model Vs JSON model

	JSON	Relational
Structure	Nested objects + arrays	Tables
Schema	Variable (and not required)	Fixed
Queries	Limited	SQL, RA
Ordering	Arrays are sorted	No
Systems	Used with programming languages and some NoSQL systems	Many commercial and open source systems

Relational Model versus JSON

	Relational	JSON
Structure	Tables	Nested Sets Arrays
Schema	Fixed in advance	"Self-describing" Flexible
Queries	Simple expressive languages	∅ widely used
Ordering	None.	Arrays.
Implementation	Native systems.	Coupled with PLs. NoSQL Systems.

XML Vs JSON.

XML versus JSON

JSON Introductio

	XML	JSON
Verbosity	More	Less
Complexity	More	Less
Validity	DTDs XSDs widely used	JSON Schema not widely used
Prog. Interface	Clunky "Impedence mismatch"	More direct
Querying	XPath - XQuery XSLT -	JSON Path JSON Query